SDN-based Big Data Caching in ISP Networks

Yong Cui, Jian Song, Minming Li, Qingmei Ren, Yangjun Zhang and Xuejun Cai

Abstract—Cooperative cache has become a promising technique to optimize the traffic by caching big data in networks. However, controlling distributed cache nodes to update cached contents synergistically is still challenging in designing cooperative cache systems. This paper proposes an SDN-based Cooperative Cache Network (SCCN) for ISP networks, aiming to minimize the content transmission latency while reducing the inter-ISP traffic. Based on the proposed increment recording mechanism, the SCCN Controller can timely capture the change of content popularity, and place the most popular contents on the appropriate SCCN Switches. We formulate the optimal content placement as a specific multi-commodity facility location problem and prove its NP-hardness. We propose a Relaxation Algorithm (RA) based on relaxation-rounding technique to solve the problem, which can achieve an approximation ratio of $1/2$ in the worst case. To solve large scale problems for big data efficiently, we further design a Heuristic Algorithm (HA), which can find a near-optimal solution with three orders of magnitude speedup compared to RA. Specifically, HA can achieve a desirable tradeoff between the transmission delay and the Internet traffic. We implement a prototype based on `Open vSwitch` to demonstrate the feasibility of SCCN. Extensive trace-based simulation results show the effectiveness of SCCN under various network conditions.

Index Terms—Big Data, Cooperative Cache, SDN, ISP Networks.

1 Introduction

Global IP traffic will pass a new milestone of 2.0 zettabytes per year in 2019 [1]. The traffic is dominated by exchanging a large amount of popular contents, e.g., photos and videos. To cope with the explosive growth of network traffic, many efforts have been performed to reduce the bandwidth usage (e.g., [2]–[5]), especially by caching the big data in networks. The P2P approach [2] is a scalable solution, but its crucial weakness is unavailability due to the unpredictable nature of users' cooperation [3]. Content Delivery Network (CDN) (e.g., [4], [6]) has been a successful business model, but its agility is limited in server deployment [7]. Moreover, due to the limited information about the network conditions, CDNs are hard to meet the requirements of traffic management from ISPs' perspective, especially to reduce the inter-ISP traffic [8].

In order to reduce the Internet traffic and improve users' experience, there is an emerging technical trend which enables network devices to be equipped with cache modules. Several devices linked together can cache and share the big data to reduce duplicates and improve performance efficiently. For example, Cisco has developed Web Cache Communication Protocol (WCCP) [9], a router-cache protocol that localizes network traffic and distributes load across multiple cache nodes. It is possible for ISPs to exploit the cooperative cache system for big data caching and delivery. However, how to control distributed cache nodes to update cached contents synergistically is still challenging in designing cooperative cache systems. To guarantee that cache nodes can detect the dynamic popularity of contents and make placement decisions efficiently, an appropriate centralized control is needed [10], [11].

Software Defined Network (SDN) [12]–[14] as a new networking paradigm can achieve logically centralized control over the distributed network nodes. Taking advantage of SDN, we propose an SDN-based Cooperative Cache Network (SCCN) for ISP networks, which employs SCCN Controller to coordinate distributed SCCN Switches (i.e., cache nodes). SCCN Switches linked together can cache and share contents with each other. SCCN Controller can detect the most popular contents at the global level, perform intensive computations of content placement decisions and populate new request forward directions to SCCN Switches quickly. Moreover, with the help of the centralized controller, the distributed caching problem can be transformed into a centralize facility location problem. Since the content placement decision is sensitive to the accurate knowledge of the content popularity, we design an increment recording mechanism based on the Least Recently Used (LRU) policy [15] to capture the popularity change of contents. Based on the request history (frequency-based) and increment records (recency-based), content placement decisions are made periodically or triggered by flash crowds, which can easily adapt to changes of content popularity over time.

We formulate the content placement as a specific multi-commodity facility location problem and prove its NP-hardness. Different from general facility location problems, we not only aim to satisfying the disk space constraints, but also jointly optimize the transmission delay and the inter-ISP traffic while satisfying the cache

• *Yong Cui, Jian Song, Qingmei Ren and Yangjun Zhang are with Tsinghua University, Beijing, P.R. China (Email: cuiyong@tsinghua.edu.cn, songj12, rqm15, zhangyangjun13@mails.tsinghua.edu.cn).*

• *Minming Li: Department of Computer Science, City University of Hong Kong, Hong Kong, P.R. China (Email: minming.li@cityu.edu.hk).*

• *Xuejun Cai: Ericsson, Sweden (Email: xuejun.cai@gmail.com).*

capacity constraints. To give a performance benchmark, we design a Relaxation Algorithm (RA) based on a relaxation-rounding technique [16] to solve the problem with an approximation ratio of $1/2$ in the worst case. Furthermore, to provide a highly efficient solution for big data case in practical large scale systems, we propose a Heuristic Algorithm (HA) based on the concept of *Circular Convex Set* [17], which can reduce the solution space from $2^n - 1$ to n for placing each content on n cache nodes. HA is not only feasible to solve large scale problems, but also allows SCCN Controller to update the placement decision more frequently, which can adapt to the change of content popularity more gracefully. Specifically, it can achieve a desirable tradeoff between the transmission delay and Internet traffic by selecting contents with different features to cache.

We implement SCCN using `Open vSwitch` [18] and `Ryu controller` [19] to show that SCCN is feasible and incurs low overhead. We also conduct extensive simulations to evaluate our algorithms against several distributed algorithms based on $14TB$ real traces collected at border routers of a campus network. The simulation results show that HA can achieve near optimal placements with three orders of magnitude speedup compared to RA under various network conditions. In summary, our key contributions are as follows:

- We design a novel SDN-based Cooperative Cache Network (SCCN) to provide a centralized control over the distributed cache nodes in ISP networks, which ensures a available cooperative cache for big data traffic. SCCN Controller can capture the popularity changes of contents based on the proposed incremental record mechanism and place contents among cache nodes optimally.
- We formulate the optimal content placement as a multi-commodity facility location problem and prove its NP-hardness, which aims to jointly optimize the transmission delay and the inter-ISP traffic. We design a Relaxation Algorithm to solve the problem with an approximation ratio of $1/2$ in the worst case. To solve large scale problems efficiently, we further design a Heuristic Algorithm (HA), which can find a near-optimal solution with three orders of magnitude speedup compared to RA.
- We implement a prototype based on `Open vSwitch` and conduct extensive simulations based on real traces to show the feasibility and effectiveness of SCCN.

The rest of this paper is organized as follows. Section II describes the design of SCCN. Section III formally defines the content placement problem. Two algorithms are presented in Section IV and V, respectively. Simulation and experimental results are presented in Section VI. Finally, we summarize the related work in Section VII and conclude the paper in Section VIII.

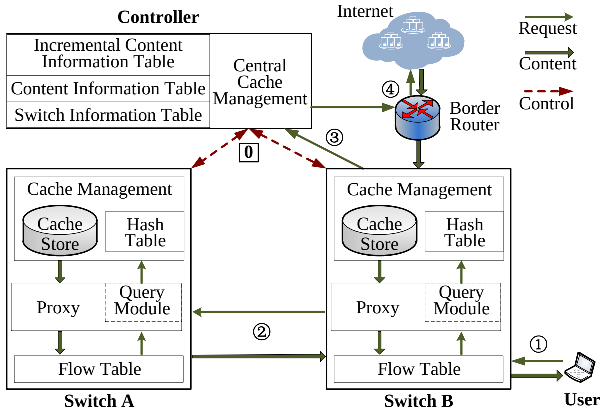


Fig. 1. The Operation Mechanism of SCCN

2 System Framework

In this section, we first outline the SCCN framework and depict the design of the main components. Then we propose a series of strategies to address practical issues for SCCN deployment in real networks.

2.1 Framework Design

SCCN performs coordinated caching in networks through two main components as shown in Fig. 1. The first is a central SCCN Controller, and the second is a set of SCCN Switches (i.e., cache nodes). The SCCN Controller has a global view of the network states. It collects the request counts from SCCN Switches and records the recent requests of contents which are not cached in the network. Based on the information, the Controller makes content placement decisions. Then the SCCN Controller updates information (e.g., the locations of contents), and installs the entries of request forward directions on SCCN Switches to indicate how to forward requests to the appropriate cache nodes. SCCN Switches based on `Open vSwitch` cache contents in accordance with the decision made by SCCN Controller, providing contents for local users. If a Switch does not cache the content that local users request, it will forward the request to other SCCN Switches under the indication of the Controller. We can deploy the content placement algorithm in the SDN controller by programming. Adding a proxy service and cache modules in SDN switches enables the implementation of SCCN.

SCCN Controller: The SCCN Controller maintains a Switch Information Table (SIT), a Content Information Table (CIT) and an Incremental Content Information Table (ICIT). SIT records profiles of every SCCN Switch, e.g., the residual capacity of cache, the interface bandwidth and so on. CIT stores content information, e.g., the location of contents, the request counts and so on. It collects the information from SCCN Switches periodically, and executes in-cache Least Frequently Used (LFU) strategy [15], i.e., the counts are defined for cached contents only. The structure of ICIT is similar to CIT. The difference is that ICIT records the information of contents which are not cached in the network, but have been requested by users recently. It executes recency/frequency-based strategy [15]. When a recorded content is hit, it is moved to the start of the list and its hit count increases by one. New requested contents are inserted at the beginning of the list, and the content at the end of the list is discarded.

The Controller makes content placement decisions periodically (e.g., every hour) according to the information recorded in SIT, CIT and ICIT. In addition, when the requested frequency of a content exceeds a threshold value within a time window, the placement update will also be triggered, which can capture the flash crowds in time, thereby avoiding the congestion of the access links. It also deals with content requests from Switches when Switches cannot find the content locations in its local Hash Table.

SCCN Switch: An SCCN Switch consists of a proxy service, a cache management service and a Flow Table (FT). The proxy service is used to take over request sessions. Cache management service maintains a Hash Table (HT) and a Cache Store (CS) to provide query and content storage services. CS stores contents, hash indexes of content requests and the version of contents. HT records hash indexes of content requests, requests count and locations, i.e., which node caches the content. FT records flow entries which contain Headers (to match content requests and other type of flows) and Actions (to tell the Switch what direction to forward the flows).

Workflow: When an SCCN Switch starts, it will connect to TCP port 6633 of the SCCN Controller and setup a TLS channel. Then a connection is established by exchanging `Hello` message between them. The system operation includes the following phases: **(1)** When SCCN Switch B receives users' flows, it redirects specific flows to local proxy service, and forwards other types of flows according to default low priority rules in FT. Proxy service calculates the hash value of the request [20], and then sends a retrieval request to the HT. If the hash value hits a record in the HT directed to the local cache, cache management service returns the content to the proxy service and updates the request count. The proxy service forwards the response to the Switch. The Switch translates source IP and port of the response to the original server's IP and port, and forwards the response to the user. **(2)** If the hash value hits a record directed to Switch A, the request will be forwarded to Switch A. Switch A as the server node will send the content to Switch B. **(3)** If the match fails, both the content request and hash value are sent to the Controller by the `Packet-in` message, which will retrieve the request in the CIT. If the request hits in the Controller's record, the Controller returns a message to add a new flow entry on the FT of Switch B. Switch B will forward the request according to the direction and records the location in the HT. **(4)** Otherwise, SCCN Controller forwards the original content request to the Internet and records the request in the ICIT. When the Controller updates the placement decision, it will update FT of the corresponding Switch by the `flow-mod` message. Using hash values to match content requests can accelerate the retrieval speed. Moreover, the hash values of the requests are all calculated on the Switches and the Controller only handles requests that failed to match in local HT, which relieve the pressure of the Controller.

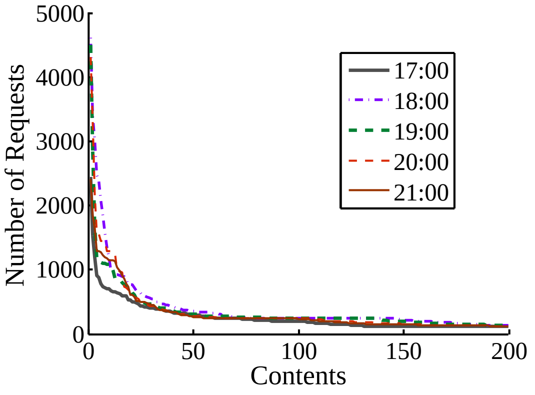


Fig. 2. Requests Count of the Top 200 Popular Contents

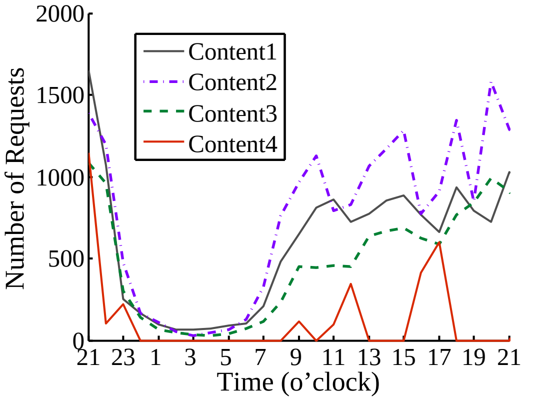


Fig. 3. Number of Requests of Contents Every Hour

2.2 Practical Considerations

A series of practical issues must be considered for SCCN design and deployment in the real networks.

2.2.1 Content Popularity Estimation

To find the popularity change of contents, we analyzed real trace data collected at border routers of a campus network. We filter the requests of the top 200 popular contents from the collected traces of each hour. The request counts of the top 200 popular contents during the peak hours (17:00-21:00) are shown in Fig. 2. The requests of more popular contents occupy larger proportion of all requests, which is coincident with the Zipf Distribute [21].

Moreover, we selected the four most popular contents at 21:00 from traffic traces, and recorded the number of requests of these contents in the following every hour as shown in Fig. 3. We find that the popularity of contents will last for a long time, but the request pattern of these contents significantly changes in every hour. This indicates that the frequency-based replacement policies are not sensitive enough to capture the change of the request pattern, especially the flash crowds. On the other hand, this indicates that employing simple recency-based replacement policies (e.g., LRU) and updating placement frequently should result in "cycling" of the cache [22]. The performance of the placement decision is sensitive to accurate knowledge of the content request pattern. We design a recency/frequency-based strategy to estimate the demand of contents. SCCN Controller makes content placement decisions based on frequency-based request history and recency-based increment recording, which can not only adapt to the change of requests timely, but also avoid caching jitter.

2.2.2 Placement Update Overhead and Frequency

The overhead of placement update mainly includes the placement decision computation and content migration between cache nodes. We design the efficient Heuristic Algorithm, which can find a near-optimal solution with three orders of magnitude speedup compared to the relaxation-based approximation algorithm. It allows us to update placement every hour, rather than to update placement every week like Lagrangian Relaxation-based Solution (LRS) [22]. To reduce the migration overhead, we design the delayed fetching mechanism. When a new placement requires node i to cache content f, i does not request f immediately, but waits until a local user requests f, then fetches it from the Internet and stores a copy. It can avoid the traffic burst of simultaneous migration.

2.2.3 Routing Selection

When Switch A cannot fetch a content locally, it can fetch the content from a remote Switch. In this case, SCCN Controller just tells it the direction, i.e., where A can fetch the content. The routing selection of content request is beyond the scope of this paper. Thus, we assume that Switch A routes the request according to the routing protocol of the network (e.g., shortest path routing) [22].

2.2.4 Cache Coherence Issue

The cached contents in SCCN are mainly multi-media contents [23], [24]. They are updated not as frequently as dynamic/personalized Web pages. Thus, we adopt adaptive TTL approach to maintain cache coherence. When a cache node fetches and stores a content, it records the store time in the CS as the version information. If the current time minus the store time exceeds a threshold value, the node will fetch the content from the Internet again. Studies [25] have shown that adaptive TTL can keep the probability of stale contents within a reasonable bound ($< 5\%$).

3 Problem Formulation

To provide a more precise description of SCCN, we model the network by a directed graph $G = (V, E)$, where V and $E = V \times V$ are the sets of SCCN Switches and directed edges. Each SCCN Switch $i \in V$ is defined as a cache node in the network. The cache size of each node i is denoted as S_i and the number of cache nodes is n. Each edge $(i, j) \in E$ in the graph represents a link from node i to node j. The contents needed to be placed are denoted by set $\mathbf{F} = \{f_1, f_2, \cdots, f_m\}$. The Controller needs to make a placement decision for m contents periodically or triggered by flash crowds. Content $f_k \in \mathbf{F}$ has size B_k. A list of symbols used in this paper is summarized in TABLE 1.

TABLE 1. Index of Symbols

Term	Definition
S_i	The cache capacity of cache node i
f_k	The content which can be cached in the network
B_k	The size of content f_k
D	The transmission delay between neighbouring nodes
$I(f_k)$	The delay for transmitting f_k from the Internet
$H_{ij}(f_k)$	The hops number for transmitting f_k from i to j
R_j	The incoming rate of content requests at node j
$r_j(f_k)$	The incoming rate of requests for f_k at node j
$p_j(f_k)$	The proportion of requests for f_k at node j
$d_{ij}(f_k)$	The delay that i obtains content f_k from j
$U_{ij}(f_k)$	The saved transmission delay when j satisfies i for f_k
$T_i(f_k)$	The traffic that satisfying the demand of node i for f_k

3.1 Transmission Delay

We define the node requesting contents as the client node, and the node providing contents as the server node. The delay of fetching a content f_k from a neighbouring node and from the Internet are denoted as D and $I(f_k)$ respectively. Requests for content f_k issued by users are first directed to the directly connected cache node i. Once receiving requests, if i caches the content, it returns the corresponding content f_k to users and the delay is considered as 0, otherwise i forwards these requests to the server node in according with the FT. We assume that the transmission delay is proportional to the number of hops between i and j as in [22]. The delay $H_{ji}(f_k)D$ is positively correlated to the hop count, where $H_{ji}(f_k)$ denotes the number of hops for transmitting f_k from j to i. If no nodes cache the content f_k, i must download it from the Internet and the corresponding delay is $I(f_k)$. We consider the Internet as a special cache node denoted as $n + 1$. The transmission delay can be defined as follows.

Definition 1. Transmission Delay *is the delay that node i obtains f_k from node j, which can be formulated as,*

$$d_{ij}(f_k) = \begin{cases} H_{ji}(f_k) \cdot D, & j = 1, \ldots, n; \\ I(f_k), & j = n+1, \end{cases}$$

where $H_{ji}(f_k)$ denotes the hops between i and j.

3.2 Time Utility and Traffic Cost

For each node j, we define the average incoming rate of content requests as R_j. The proportion of requests for contents at j is represented by the vector $\overrightarrow{p_j} = (p_j(f_1), \cdots, p_j(f_m))$. The rate of requests for f_k at j is denoted as $r_j(f_k)$, which can be calculated by $r_j(f_k) = p_j(f_k) \cdot R_j$. Generally speaking, fetching contents from cache nodes will suffer less delay than from the Internet, which is defined as *Time Utility*.

Definition 2. Time Utility *is the saved transmission delay when node j satisfies node i's demand for content f_k, which can be formulated as,*

$$U_{ij}(f_k) = r_i(f_k) \cdot [I(f_k) - d_{ij}(f_k)], \tag{1}$$

where $i = 1, \cdots, n$ and $j = 1, \cdots, n+1$.

Note that $U_{i(n+1)}(f_k) = 0$, i.e., the *Time Utility* is 0 when the Internet satisfies node i's demand, which means that no transmission delay is saved.

Definition 3. Traffic Cost *is the total data traffic generated by satisfying the demand of node i for content f_k, which can be formulated as $T_i(f_k) = r_i(f_k) \cdot B_k$.*

Since local cache and cooperative cache between nodes will not generate Internet traffic, the Internet traffic cost is only caused by downloading the contents from the Internet because no node caches the contents.

3.3 Decision Vector

The content placement decision is influenced by the content request distribution. We define the fetch decision vector and placement decision vector to depict the cooperative caching. *Fetch Decision Vector* $\overrightarrow{x_i}(f_k) = (x_{i1}(f_k), \cdots, x_{i(n+1)}(f_k))$ decides where node i fetches content f_k as

$$x_{ij}(f_k) = \begin{cases} 1, & \text{node } i \text{ will fetch } f_k \text{ from } j; \\ 0, & \text{otherwise,} \end{cases}$$

where $i = 1, \cdots, n$, $j = 1, \cdots, n+1$ and $k = 1, \cdots, m$. Especially, $x_{i(n+1)}(f_k) = 1$ indicates that node i fetches f_k from the Internet. *Placement Decision Vector* $\overrightarrow{y}(f_k) = (y_1(f_k), \cdots, y_n(f_k))$ decides where to place content f_k,

$$y_j(f_k) = \begin{cases} 1, & \text{node } j \text{ caches } f_k; \\ 0, & \text{otherwise,} \end{cases}$$

where $j = 1, \cdots, n$ and $k = 1, \cdots, m$. There may be more than one element equal to 1, which means that f_k is cached at more than one node. In addition, if all elements are 0, it shows that f_k is not cached in the whole network. Especially, $y_{n+1}(f_k) = 1$ means the Internet stores f_k.

3.4 Model Description

The content placement decision aims to maximize the network-wide *Time Utility* with the capacity constraints of cache nodes and the Internet *Traffic Cost* constraints, i.e.,

$$\max \sum_{f_k \in \mathbf{F}} \sum_{i \in V} \sum_{j=1}^{n+1} U_{ij}(f_k) \cdot x_{ij}(f_k) \tag{2}$$

s.t.

$$\sum_{j=1}^{n+1} x_{ij}(f_k) = 1, \qquad \forall i \in V, f_k \in \mathbf{F}, \tag{3}$$

$$x_{ij}(f_k) \le y_j(f_k), \qquad \forall i, j \in V, f_k \in \mathbf{F}, \tag{4}$$

$$\sum_{f_k \in \mathbf{F}} B_k \cdot y_j(f_k) \le S_j, \qquad \forall j \in V, \tag{5}$$

$$\sum_{f_k \in \mathbf{F}} \sum_{i \in V} T_i(f_k) \cdot x_{ij}(f_k) \le \rho \cdot \Psi, j = n+1, \tag{6}$$

$$x_{ij}(f_k), y_j(f_k) \in \{0,1\}, \qquad \forall i \in V, f_k \in \mathbf{F}, \; j = 1, \cdots, n+1. \tag{7}$$

The objective (2) is to maximize the network-wide *Time Utility*. To satisfy all the requests, constraints (3) ensure every request from node i can fetch content f_k from only one place. Constraints (4) require that node j has cached f_k when other nodes request f_k from it. Constraints (5) establish the finite capacity of each node. Constraint (6) provides the Internet *Traffic Cost* limitation, where Ψ is the capacity of the access link. We can control the traffic proportion by adjusting the coefficient $\rho \in [0, 1]$. Note that, if the traffic constraint is too strict to get a feasible solution, the network administrator should add the capacity of switches or increase ρ. We will analyze how to choose a reasonable ρ in our simulation. Finally, constraints (7) provide the nonnegativity and integrality of the decision variables. By reducing from the knapsack problem, we have the following theorem:

Theorem 1. *Solving the optimization problem presented in (2) is NP-hard.*

Proof: We transform 0-1 knapsack problem which is NP-hard to our problem presented in (2) as follows. Given a set of m items denoted by set $\mathbf{F} = \{f_1, f_2, \cdots, f_m\}$ and a knapsack j with capacity S. Denote the profit and weight of one item as $\mathcal{U}(f_k)$ and $\mathcal{S}(f_k)$ respectively. The problem is to select the most valuable set of items to maximize the profit, satisfying the capacity constraint S.

We define the cache space required by caching content f_k as the weight, i.e., $\mathcal{S}(f_k) = B_k$, and the saved transmission delay as profit, i.e., $\mathcal{U}(f_k) = \sum_{i \in V} U_{ij}(f_k) \cdot x_{ij}(f_k)$. And $y_j(f_k) = 0$ means that content f_k is not cached in j. Then, the 0-1 knapsack problem is equivalent to the problem where only one cache node exists in the network. This completes the proof. $\square$

4 Approximation Algorithm Based on Relaxation-Rounding

To handle the problem in (2), we design the Relaxation Algorithm (RA) based on a relaxation-rounding technique [16], and prove that it is at least 1/2 of the theoretical optimal scheme. The main idea is to address the original integer optimization problem by solving another relaxed problem. We first relax the 0-1 binary variables i.e., the *Fetch Decision Vector* and *Placement Decision Vector*, to real number variables ranging from 0 to 1. Intuitively, the relaxation of the *Fetch Decision Vector* to real number variable means that the users can fetch a fraction of a file from a cache node, and relaxing the *Placement Decision Vector* means that we can store a fraction of a file on a cache node. The induced relaxed problem can be solved in polynomial time. With the global optimal solution of the relaxed problem, we use a rounding procedure to generate the approximate solution to the original problem.

4.1 Relaxation

We first relax the original problem by introducing variables $\widehat{x}_i(f_k), \widehat{y}(f_k) \in [0,1]$, which substitute for the *Fetch Decision Vector* and *Placement Decision Vector* in (2). The relaxed problem is expressed as (8). As all the constraints are linear equations, the fractional optimal solution can be found in polynomial time, which is denoted by U_r. With the global optimal solution of the relaxed problem, we use a rounding procedure to generate the approximate solution to the original problem. Each element in $\overrightarrow{\widehat{x}_i}(f_k)$ denotes the tendency where node i decides to fetch content f_k, and each element in $\overrightarrow{\widehat{y}}(f_k)$ denotes the tendency about where to cache content f_k.

$$\max \sum_{f_k \in \mathbf{F}} \sum_{i \in V} \sum_{j=1}^{n+1} U_{ij}(f_k) \cdot \widehat{x}_{ij}(f_k) \tag{8}$$

subject to:

$$\begin{aligned}
&\sum_{j=1}^{n+1} \widehat{x}_{ij}(f_k) = 1, && \forall i \in V, f_k \in \mathbf{F},\\
&\widehat{x}_{ij}(f_k) \le \widehat{y}_j(f_k), && \forall i,j \in V, f_k \in \mathbf{F},\\
&\sum_{f_k \in \mathbf{F}} B_k \cdot \widehat{y}_j(f_k) \le S_j && \forall j \in V,\\
&\sum_{f_k \in \mathbf{F}} \sum_{i \in V} T_i(f_k) \cdot \widehat{x}_{i(n+1)}(f_k) \le \rho \cdot \Psi &&\\
&\widehat{x}_{ij}(f_k), \widehat{y}_j(f_k) \in [0,1], && \forall i \in V, f_k \in \mathbf{F},\\
& && j = 1, \cdots, n+1.
\end{aligned}$$

4.2 Rounding

We use the rounding technique proposed in [16] to obtain integral assignment matrices $\overrightarrow{x_i}(f_k)$ and $\overrightarrow{y}(f_k)$, replacing the fractional optimal solution. The main idea is to construct a weighted bipartite graph for every content f_k based on the fractional solution and find the maximum weighted matching to generate the solution of (2). Let $\ddot{G} = (\ddot{A}, \ddot{V}, \ddot{E}, \ddot{W}(\ddot{E}))$ denote the bipartite graph, where $\ddot{A}$ and $\ddot{V}$ are two groups of nonadjacent nodes and $\ddot{E}$ denotes the set of edges. $\ddot{A}$ and $\ddot{V}$ represent client nodes and server nodes, respectively.

First, we construct the client node set $\ddot{A} = \{\ddot{a_1} \cdots, \ddot{a_n}\}$ according to non-increasing sequence of their requesting rates $r_i(f_k)$. Note that when computing $\widehat{x}_{ij}(f_k)$, we assume that client i will request f_k from server j as long as $\widehat{x}_{ij}(f_k) > 0$. Under this assumption, each client node is able to request f_k from multiple server nodes. Although it is not the real case, the estimated results can be used to predict the contribution of each request to the global *Time Utility*.

Then, we create the nodes in $\ddot{V}$ based on $\overrightarrow{\widehat{x}_i}(f_k)$. Let $c_j = \lceil \sum_{i \in E} \widehat{x}_{ij}(f_k) \rceil$. Each server node j corresponds to c_j nodes in $\ddot{V}$, denoted by $\ddot{V} = \{\ddot{v}_{j,c} : j = 1, \cdots, n, c = 1, \cdots, c_j\}$. With $\ddot{A}$ and $\ddot{V}$, we can set up edges between them. Edges of graph $\ddot{G}$ will correspond to client and server pairs (i,j) such that $\widehat{x}_{ij}(f_k) > 0$. For each server node j, if $c_j \le 1$, there is only one node $\ddot{v}_{j,1}$ in $\ddot{V}$. For each client node i satisfying $\widehat{x}_{ij}(f_k) > 0$, add edge $\ddot{e}_{i,j,1}$ to $\ddot{E}$ and set $\ddot{w}(\ddot{e}_{i,j,1}) = \widehat{x}_{ij}(f_k)$. Otherwise, if $c_j > 1$, there are multiple nodes $\{\ddot{v}_{j,1}, \cdots, \ddot{v}_{j,c_j}\} \subseteq \ddot{V}$ corresponding to j. Let i_1 denote the minimum number satisfying $\sum_{i=1}^{i_1} \widehat{x}_{ij}(f_k) \ge 1$. For $i \in \{1, \cdots, i_1 - 1\}$, add edge $\ddot{e}_{i,j,1}$ to $\ddot{E}$ and set its weight $\ddot{w}(\ddot{e}_{i,j,1}) = \widehat{x}_{ij}(f_k)$; for i_1, add edge $\ddot{e}_{i_1,j,1}$ and set its weight $\ddot{w}(\ddot{e}_{i,j,1}) = 1 - \sum_{i=1}^{i_1-1} \ddot{w}(\ddot{e}_{i,j,1})$. This ensures that the sum of the components of $\ddot{w}(\ddot{e})$ for edges incident to $\ddot{v}_{j,1}$ is 1. If $\sum_{i=1}^{i_1} \widehat{x}_{ij}(f_k) > 1$, add edge $\ddot{e}_{i,j,2}$ and set its weight $\ddot{w}(\ddot{e}_{i,j,2}) = \sum_{i=1}^{i_1} \widehat{x}_{ij}(f_k) - 1$. We then proceed with client nodes $i > i_1$, and construct edges incident to $\ddot{v}_{j,2}$, until a total of exactly one client node is assigned to $\ddot{v}_{j,2}$, and so forth.

After constructing a bipartite graph $\ddot{G}$ for every content f_k, we process $\ddot{G}$ according to the *Time Utility* in descending order. With the constraints of node capacity and *Traffic Cost*, we perform maximum weighted matching on the bipartite graph. Let $\ddot{E}_M$ denote the results of matching. For each selected edge in the set, set $x_{ij}(f_k)$ to 1; then, all the unmodified elements in $\overrightarrow{x_i}(f_k)$ are set to 0. Thus, we obtain the solution of (2). RA consists of three steps as shown in Algorithm 1.

```
Algorithm 1: RA
Input: $\{B_1, \cdots, B_m\}$, $\{I(f_1), \cdots, I(f_m)\}$, $\{S_1, \cdots, S_n\}$,
    $r_i(f_k), H_{ij}(f_k)$ where $i, j = 1, \cdots, n$ and $k = 1, \cdots, m$.
Output: $\overrightarrow{y}(f_k)$, $\overrightarrow{x_i}(f_k)$ and $U_a$.
1: Obtain the fractional solution $\overrightarrow{\widehat{y}}(f_k)$, $\overrightarrow{\widehat{x}_i}(f_k)$ and $U_r$
   by solving a relaxed optimization.
2: Obtain the integral solution $\overrightarrow{y}(f_k)$, $\overrightarrow{x_i}(f_k)$ and $U_a$ by
   a rounding process.
3: Assign the contents to appropriate cache nodes.
4: return $\overrightarrow{y}(f_k)$, $\overrightarrow{x_i}(f_k)$ and $U_a$.
```

Let U_a denote the solution obtained by RA and U^*

denote the optimal integer solution of (2). We have the following theorem to estimate the lower bound of U_a.

Theorem 2. $U_a \geq \frac{1}{2}U^*$.

Proof: We denote the sum of *Time Utility* placed in the network as $U_r = \sum_{k=1}^{R} U_{f_k}$, where $f_k \in \mathbf{F}$ and $R \leq m$. In a similar way, U_a is expressed as $U_a = \sum_{k=1}^{R} U'_{f_k}$. $U'_{f_k} = 0$ means that f_k is not placed in the scheme of U_a. We construct a complementary solution U_c of U_a in accordance with [16], i.e., set $x_{ij}(f_k)$ to 1 for each edge in the set $\ddot{E} - \ddot{E}_M$. Then, we have $U_c = \sum_{k=1}^{R} U''_{f_k}$, where $f_k \in \mathbf{F}$ and $R \leq m$. The U''_{f_k} can be calculated as,

$$U''_{f_i} = \begin{cases} 0, & \text{if } U'_{f_i} \geq U_{f_i}, \\ U_{f_i} - U'_{f_i}, & \text{if } U'_{f_i} < U_{f_i}, \\ U_{f_i}, & \text{if } U'_{f_i} = 0. \end{cases}$$

According to [16], the rounding result satisfies $U_a + U_c \geq U_r$. It is obvious that $U_a \geq U_c$. Therefore, we have $U_a \geq \frac{1}{2}U_r \geq \frac{1}{2}U^*$. This completes the proof. □

5 Heuristic Algorithm for Content Placement

Although RA achieves a polynomial-time complexity, the complexity grows significantly with the number of contents m. It is not efficient enough for large scale problems. To ease the computational complexity of the system with larger number of contents m and cache nodes n, we design a Heuristic Algorithm (HA) to address the problem efficiently based on the concept of *Circular Convex Set* [17].

We first consider the placement of only one content f_k. Deciding where to place f_k and which clients fetch f_k from the location is equivalent to a discrete median problem [17]. Every partition of the set V induces a feasible solution. We define *Circular Convex Sets* for every content, which can reduce the number for traversing these partitions from $2^n - 1$ to n.

Definition 4. *A set $P_j \subset V$ is* Circular Convex Set $\Longleftrightarrow$ $\forall i \in P_j, \forall t \in V - P_j, U_{tj}(f_k) < U_{ij}(f_k)$. *The* unit utility *of the* Circular Convex Set *can be denoted as* $\frac{\sum_{i\in P_j} U_{ij}(f_k)}{B_k}$.

For each server node $j \in V$ we set $U_{i_1 j}(f_k) \geq U_{i_2 j}(f_k) \geq \cdots \geq U_{i_n j}(f_k)$, where $\{i_1, \cdots i_n\} = V$. Then we sequentially generate *Circular Convex Sets* for which j is a center: $\{i_1\}, \{i_1, i_2\}, \cdots, \{i_1, \cdots, i_n\}$. Hence the number of *Circular Convex Sets* centered in j is n. It is possible to traverse these partitions in polynomial-time. Then, we sequentially find the subset for all contents $f_k \in \mathbf{F}$ and server nodes $j \in V$, which can achieve the maximum *Time Utility* subject to the space constraints and traffic constraints. This subset is recorded in $COVER(f_k)$ and removed from $V(f_k)$. HA is processed iteratively until finding optimal partitions for every content, which is shown in Algorithm 2.

When the algorithm is triggered, for every content f_k, the algorithm sets an uncovered set $V(f_k)$ and a covered set $COVER(f_k)$. Line 3 sorts client nodes in descending order according to the *Time Utility* with server node j, and constructs $V'_j(f_k)$ for every server node j and every f_k. The time complexity of this step is $O(mn^2 \log_2 n)$, where m and n denote the number of contents and the number of cache nodes in the network respectively. Line 4 calculates the sum of *Time Utility* for every *Circular Convex Set*, and finds the *Circular Convex Set* achieving maximum utility and the corresponding service node j^*. The time complexity of this step is $O(mn^2 M)$, where M denotes the number of contents that network nodes can cache. Line 5 adds the *Circular Convex Set* with maximum utility to $COVER(f_k)$. Line 6 removes the subset from $V'_j(f_k)$. The algorithm will be terminated if all contents have been processed or it cannot find a cover set satisfying both capacity constraints and traffic constraints, as shown in line 8. In general, M is much larger than n, so the time complexity of Algorithm 2 is $O(mn^2 M)$.

Algorithm 2: HA

```
Input: $\{B_1, \cdots, B_m\}$, $\{I(f_1), \cdots, I(f_m)\}$, $\{S_1, \cdots, S_n\}$,
    $r_i(f_k), H_{ij}(f_k)$ where $i, j = 1, \cdots, n$ and $k = 1, \cdots, m$.
Output: $\overrightarrow{y}(f_k)$, $\overrightarrow{x_i}(f_k)$ and $U_h$
1: $\forall f_k \in \mathbf{F}$, set $COVER(f_k) = \emptyset$, $V(f_k) = V$, $C_k = |V(f_k)|$.
2: repeat
3:     $\forall j \in V, \forall f_k \in \mathbf{F}$, generate convex sets of $V(f_k)$, then by sorting these convex sets by non-increasing unit utility, we get sets $V'_{j,1}(f_k), \ldots, V'_{j,C_k}(f_k)$ .
4:     For all unselected couples $\{j, f_k\}$ and $c \in \{1, \ldots, C_k\}$ find $Q = \{i_1, \cdots, i_l\}$ such that
       $\frac{\sum_{i\in q} U_{ij^*}(f_k)}{B_k} = \arg\max_{V'_{j,c}(f_k)\neq\varnothing} \frac{\sum_{i\in V'_{j,c}(f_k)} U_{ij}(f_k)}{B_k}$
       satisfying capacity constraint of $j^*$ and Traffic Cost constraint of the Internet.
5:     $COVER(f_k) = COVER(f_k) \cup \{\{i_1, \cdots, i_l\}\}$.
6:     $V(f_k) = V(f_k) - \{i_1, \cdots, i_l\}$, update $C_k = |V(f_k)|$.
7:     $\forall i \in \{i_1, \cdots, i_l\}$, $x_{ij^*}(f_k) = 1$ and $y_{j^*}(f_k) = 1$.
8: until $V(f_k) = \emptyset$ or $q$ is not found.
9: return $\overrightarrow{y}(f_k)$, $\overrightarrow{x_i}(f_k)$ and $U_h$.
```

6 Performance Evaluation

We evaluate the effectiveness of SCCN algorithms by extensive trace-based simulation, and demonstrate the feasibility of the system by a prototype implementation.

6.1 Trace-based Simulation

To evaluate SCCN, we develop a protocol independent simulator, which can define arbitrary network topology, simulate the decision-making of SCCN Controller and interactions between Switches. We evaluate the performance of SCCN on users' *Time Utility* and the Internet traffic using real traffic traces. On the practical side, we conduct extensive simulations on a hierarchical topology of a campus network and a backbone network topology as shown in Fig. 4. Table. 2 summarizes important metrics of these two networks. Specifically, the node number and the edge number are the basic property of any topology. Besides, we also focus on their radius and diameter, because they will affect the complexity of the decision-making.

First, we analyze the behaviors of each algorithm at the same system settings and network conditions. Second, we show the impact of system settings on the performance of SCCN, i.e., how the cache capacity of nodes and the traffic constraints affect the network-wide *Time Utility*. Third, we present how different network conditions affect the performance of SCCN, such as the request pattern, the Internet transmission delay and so on. In order to evaluate the average performance, we run 10 groups of tests under different conditions. The mean and the standard deviations are used in these error-bar figures, Fig.6, 8, 10, 11. Moreover, the performance of HA is evaluated with large scale of problems.

6.1.1 Data Sets

Our evaluation is based on real traces collected at border routers of a campus network. Note that the total amount of traffic in each trace is approximately correlated to the number of host IP addresses in the sub-domain. Typical traffic volume is about $6.9GB - 178GB/hour$. The wide gap is due to the traffic demand variance between the busy hour and the idle hour. The total volume of traffic analyzed is about $14TB$, which was collected continuously in one week.

To find the solution of RA within a reasonable amount of time, we filter the requests of the top 200 popular contents from the collected traces of each hour, which are sufficient to evaluate the performance of each algorithm. Based on the traces, we set the transmission delays between neighbouring nodes to be $1ms$ and $2ms$ in hierarchical topology and backbone network topology, respectively. The size of contents are mapped to four different lengths: 1,5,10 and 20 in accordance with [22]. Moreover, the video streams from Youku [26] are used to evaluate the performance of HA with large problem sizes in real environment. They are obtained by reconstruct TCP flows and filter out the video streaming flows. The contents (i.e., video chunks) can be distinguished by the VIDs which are contained in request URLs. The number of contents is more than 11000, and the total number of requests is more than 100000. We can obtain the content size from the content length domain of HTTP response header, and the transmission delay from the

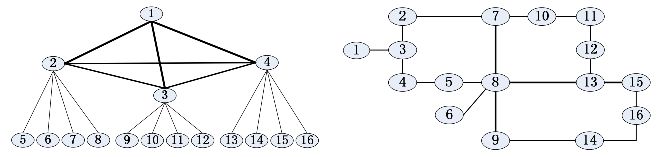

(a) Hierarchical Network Topology (b) Backbone Network Topology

Fig. 4. The Network Topologies Used for Evaluation

time stamp. An example of request distributions and content sizes is shown in TABLE 3. Note that the larger the number of a content is requested, the smaller its size is. This indicates that the Youku server has taken some optimization strategies for video transmission. Because the beginning part of videos is requested more times by users, Youku server cuts the beginning part into smaller chunks. This strategy can shorten the start time of videos.

6.1.2 Performance Metrics

We define Acceleration Ratio (A_{ratio}) and Traffic Ratio (T_{ratio}) as the performance measures. Acceleration Ratio denotes the ratio between the saved transmission delay and the original Internet delay, and Traffic Ratio denotes the ratio between the saved traffic and desired traffic without SCCN. They are formulated as,

$$A_{ratio} = \frac{\sum_{f_k \in \mathbf{F}} \sum_{i \in V} \sum_{j=1}^{n+1} U_{ij}(f_k) \cdot x_{ij}(f_k)}{\sum_{f_k \in \mathbf{F}} \sum_{i \in V} r_i(f_k) \cdot I(f_k)}, \quad (9)$$

$$T_{ratio} = \frac{\sum_{f_k \in \mathbf{F}} \sum_{i \in V} [T_i(f_k) - T_i(f_k) \cdot x_{i(n+1)}(f_k)]}{\sum_{f_k \in \mathbf{F}} \sum_{i \in V} T_i(f_k)}. \quad (10)$$

The fractional optimal solution is presented as a reference bound (RB). Moreover, we compare RA and HA with two distributed algorithms proposed in [4], i.e., Local-Greedy Algorithm (LG) and Local-Greedy-Gen Algorithm (LGG). LG adopts full replication strategy, where each node caches the K most popular contents for itself, and nodes do not share contents with each other. On the contrary, LGG employs no replication strategy, where only a single copy of each content is cached and nodes share contents with each other.

6.1.3 **Algorithm Behaviors**

This part aims to distinguish the characteristic of RA, HA, LG, LGG and MIP [22]. The Cumulative Acceleration Ratio of the top 200 popular contents and their duplicates is shown in Fig. 5. Due to the similar results in hierarchical topology, we only present the results in

TABLE 2. The Metrics of The Two Network Topologies

Topology	$\lvert V\rvert$	$\lvert E\rvert$	*Radius*	*Diameter*
Hierarchy	16	18	$2hops$	$3hops$
Backbone	16	18	$4hops$	$7hops$

TABLE 3. Request Distributions of Video Contents

Content	f_1	f_2	f_3	f_4	f_5
Size	$530KB$	$2.9MB$	$2.5MB$	$7.8MB$	$16.8MB$
RequestCount	8811	1304	1005	449	184

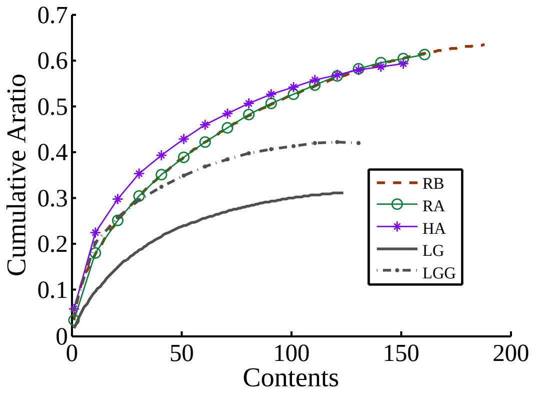
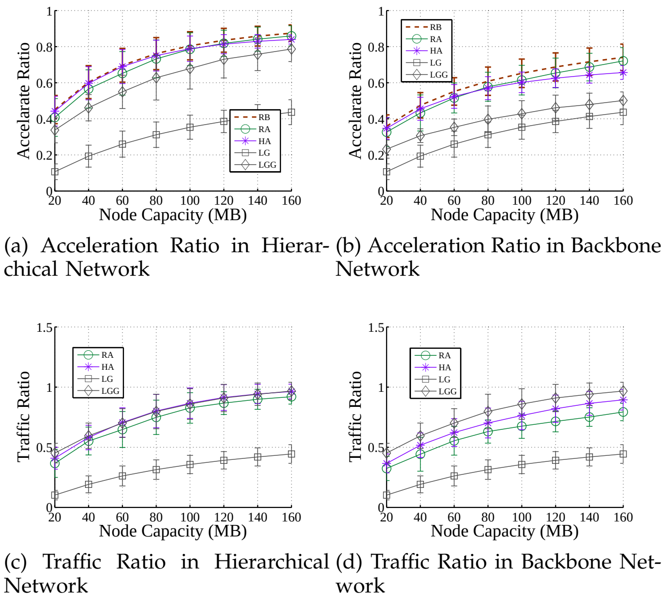


Fig. 5. Cumulative Acceleration Ratio of Contents in Backbone Network

backbone topology. X-coordinate represents the number of contents and their duplicates cached in the network, and Y-coordinate represents the sum of Acceleration Ratio contributed by these contents. Note that RA and HA achieve larger *Time Utility* by placing more contents and duplicates. Because multiple duplicates reduce the *Time Utility* of single duplicate, the Cumulative Acceleration Ratio of LGG increases quickly when placing the most popular contents by employing no replication strategy. Although LGG employs no replication strategy, it caches the most popular contents regardless their sizes, causing that it caches less number of contents than RA and HA. Moreover, LGG cannot exclude the client nodes with negative *Time Utility*, causing the Cumulative Acceleration Ratio to decrease when placing the less popular contents. The full replication strategy makes the Cumulative Acceleration Ratio of LG always lower than other algorithms described above.

To distinguish the computational complexities of our algorithms, we run these algorithms on a regular desktop with dual-core Intel i3-3220 3.3GHz CPU and 8G DDR3 memory. The running time of each algorithm at the same system setting is shown in TABLE 4. Note that the high efficiency of HA enables it to solve large scale problems and allow SCCN Controller to update the placement decisions more frequently. Moreover, to depict the overhead of SCCN Controller, we count the total number of requests and the number of requests which SCCN Controller handles under the solutions of HA as shown in TABLE 5. Since the Controller only needs to handle requests that failed to match in local HT, the number is less than 3.5% of the total number of requests.



(a) Acceleration Ratio in Hierarchical Network (b) Acceleration Ratio in Backbone Network

(c) Traffic Ratio in Hierarchical Network (d) Traffic Ratio in Backbone Network

Fig. 6. Acceleration Ratio and Traffic Ratio with Different Cache Size

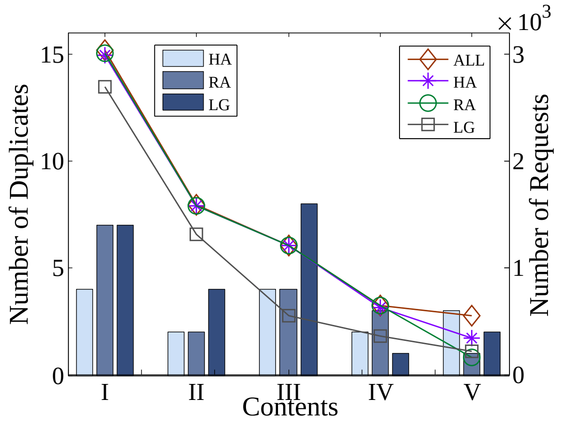


Fig. 7. The Number of Duplicates of the Top 5 Popular Contents in Backbone Network

6.1.4 *Impact of Cache Size*

This simulation illustrates how the cache capacity impacts the *Time Utility* and *Traffic Cost*. Since we have observed similar results under circumstances where cache nodes have equal capacity or heterogeneous capacities, we only show results where cache nodes have equal capacity. The cache size of each node is set to increase from 20 to 160 with step length 20. We present the Acceleration Ratio and the Traffic Ratio in Fig. 6.

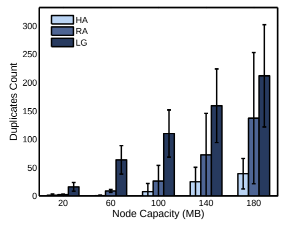
A notable phenomenon is that the Acceleration Ratio

TABLE 4. Computing Time of Each Algorithm

Scheduler	*LRS*	*RA*	*HA*	*LG*	*LGG*
Time	$2h$	$1.2h$	$1.9s$	$0.8s$	$0.8s$

TABLE 5. Controller Overhead of HA

Time	17:00	18:00	19:00	20:00	21:00
Total	52090	79949	67240	73459	61185
Controller	1085	1218	1427	1482	1352


(a) The Number of Duplicates in Hierarchical Network

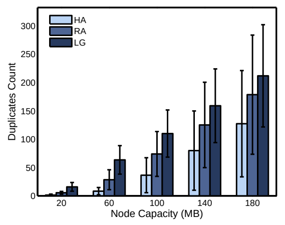
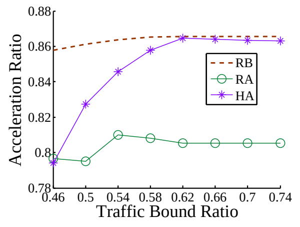

(b) The Number of Duplicates in Backbone Network

Fig. 8. The Number of Duplicates with Different Cache Size


(a) Acceleration Ratio in Hierarchical Network

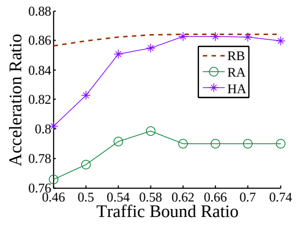
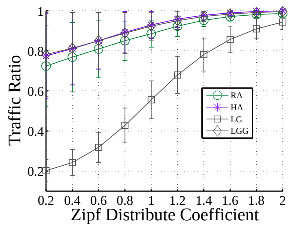

(b) Acceleration Ratio in Backbone Network

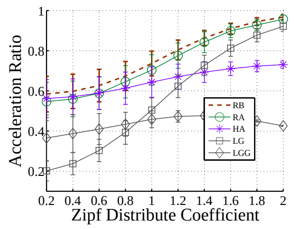
Fig. 9. Acceleration Ratio and Traffic Ratio with Different Internet Traffic Constraints

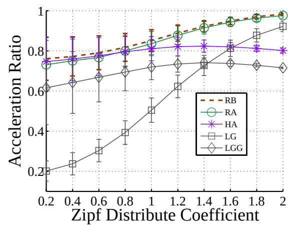
and Traffic Ratio grow quickly when cache capacity is small and then grow slowly with cache capacity increasing. The reason is that the requests of most popular contents occupy the majority of all requests, and nodes can obtain less *Time Utility* by caching less popular contents with increased capacity. Nodes do not share contents with each other under LG, which reduces the least the inter-ISP traffic, whereas it can save more in-network traffic. The total number of duplicates is shown in Fig. 8. The number of duplicates of the top 5 popular contents in Backbone, and their number of requests are shown in Fig. 7. RA can increase the Acceleration Ratio further by placing more duplicates with the cache capacity increasing, but this makes the Traffic Ratio of RA lower than HA.

6.1.5 *Impact of Internet Traffic Constraint*

To show the effect of the traffic constraints as shown in (6), we present the Acceleration Ratio with cache capacity 100 of each node in Fig. 9. When coefficient ρ increases to 46% at the current system setting, HA and RA can obtain the feasible solutions. The Acceleration Ratios of HA and RA increase with relaxation of the Internet Traffic Cost constraints. Note that the deviation of RA generated by rounding causes the fluctuation of results. However, HA can not only achieve a tradeoff between the cache capacity and Internet traffic, but also achieve a tradeoff between the transmission delay and Internet traffic by selecting contents with different features to cache as shown in Fig. 6 and Fig. 9.


(a) Acceleration Ratio in Hierarchical Network


(b) Acceleration Ratio in Backbone Network


(c) Traffic Ratio in Hierarchical Network

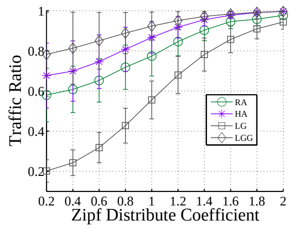

(d) Traffic Ratio in Backbone Network

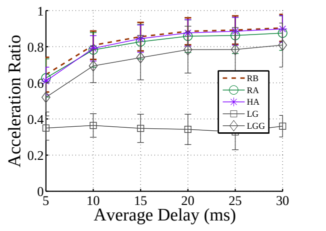
Fig. 10. Acceleration Ratio and Traffic Ratio Affected by Request Pattern

6.1.6 *Impact of Request Pattern*

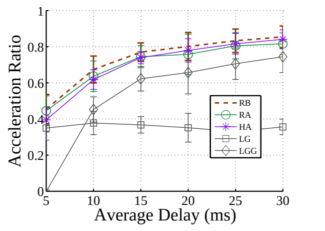
The performance of SCCN under different request patterns with cache capacity 120 of each cache node is shown in Fig. 10. Larger Zipf Distribute Coefficient means more requests of the most popular contents, i.e., the requests of most popular contents occupy larger proportion of all requests. We set the Zipf Distribute Coefficient α to increase from 0.2 to 2 in step 0.2. The Acceleration Ratios of RA and LG grow quickly with the Zipf Distribute Coefficient α increasing. That is because RA and LG can place more duplicates of the most popular contents, which can achieve more *Time Utility* than placing less popular contents. The Traffic Ratio of RA and LG is worse than HA and LGG, because duplicates take up more cache space and users must fetch contents with less popularity from the Internet. However, when $\alpha = 1$, which has been observed in real web access traces [21] and is consistent with our data trace, HA and RA can achieve 64%, 69% Acceleration Ratio respectively, which are better than LG and LGG.

6.1.7 *Impact of Average Internet Delay*

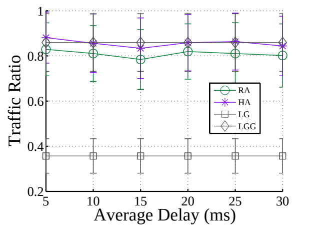
We show how Acceleration Ratio and Traffic Ratio change along with the different Internet transmission delays in Fig. 11. We assume that the fetch delay distribution of 200 contents follows Gauss distribution with the average delay varying from $5ms$ to $30ms$ in step $5ms$, keeping distribution variance at $5ms$. All the cache nodes are configured with 120 cache capacity. Acceleration Ratio increases with Average Delay rising except LG, because nodes do not share contents with each other in LG. Both HA and RA have a good performance, which is reasonable according to formula (1). When the Internet



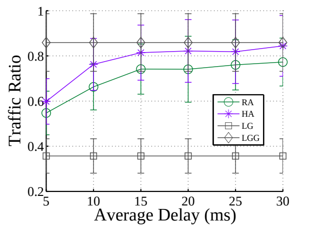
(a) Acceleration Ratio in Hierarchical Network



(b) Acceleration Ratio in Backbone Network

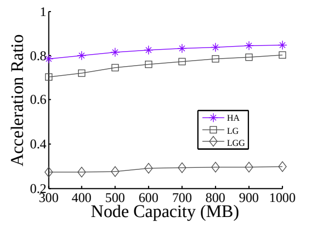


(c) Traffic Ratio in Hierarchical Network

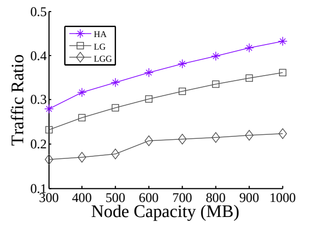


(d) Traffic Ratio in Backbone Network

Fig. 11. Acceleration Ratio and Traffic Ratio Affected by Average Internet Delay



(a) Acceleration Ratio in Backbone Network



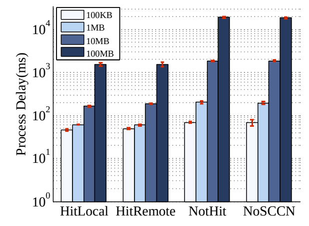
(b) Traffic Ratio in Backbone Network

Fig. 12. The Performance of HA

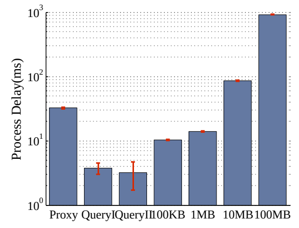
transmission delay is small, LGG cannot accelerate the network any more. On the contrary, HA can choose a suitable cover set and replace duplicates to avoid getting negative utility gain.

6.1.8 The Performance of HA with Large Problem Sizes

To illustrate the effectiveness of HA, we evaluate HA with large problem sizes in real network conditions. The cache size of each node is set to increase from $300MB$ to $1GB$. We present the Acceleration Ratio and the Traffic Ratio in backbone network topology in Fig. 12. The Acceleration Ratio in hierarchical topology is higher than that in backbone network topology. Due to space limitation, we do not present the performance of HA in hierarchical topology. Note that the Traffic Ratio of HA is better than LGG. The volumes of contents with the larger requests number are smaller than contents with less popularity, which reduces the Traffic Ratio of LGG. However, HA can select cached contents considering the cache capacity and Internet traffic synthetically. Moreover, the performance of LG is better than LGG, which



(a) Average Processing Time under four cases



(b) Processing Time of Download, Proxy and Query

Fig. 13. Operating Efficiency of SCCN

is because that the Request Distribution Coefficient α of video trace of one day is larger than one hour. It is consistent with the conclusion drawn in Fig. 10.

6.2 Prototype Implementation

In this section, we implemented a prototype on top of Mininet [27] to validate the feasibility of SCCN. We emulate an SDN network with `Open vSwitch v2.3.0` [18] and `Ryu controller` [19]. To handle users' requests, we deploy a proxy service and a cache management service at SCCN Switch side. At the SCCN Controller side, we deploy a central cache management service to manage global information of caches.

Proxy and Cache Management Services: The proxy service is implemented by adding components to `mitmproxy v0.12` [28] to take over request sessions. Both cache management and central cache management service are implemented by adding components to `Django v1.8` [29]. Cache management service maintains HT and CS on the Switch to provide query and content storage services. Users' requests will be sent to central cache management service when match failed in local cache management service. Central cache management service maintains SIT, CIT and ICIT on the Controller, which has a global view of all the cache storage and cache requests distribution.

Experiment Analysis: We conduct an evaluation based on the prototype on a hierarchical topology of a campus network. We recorded the average processing time for 100 requests under four cases in Fig. 13a, i.e., hitting a record locally (Hit Local), hitting a record on remote Switch (Hit Remote), not hitting in the network (Not Hit) and without SCCN (No SCCN). Note that SCCN performs a little efficiently compared with No SCCN when processing the requests for small contents, due to the overhead of redirection and querying. However, SCCN is much more efficient for large content requests, because the bandwidth and transmission delay of intranetwork is much better than accessing Internet.

To understand the operating efficiency of SCCN, we show the processing time of proxy, query locally (Query I), query on remote Switch (Query II), and the download time for 100KB, 1MB, 10MB and 100MB in Fig. 13b. Note that SCCN introduces about 30ms additional delay due to the proxy, which is responsible for the request forward and the proxy query. However it is efficient enough for all contents, especially when the contents are larger than 10MB. Moreover, the number of requests that the Controller needs to handle is less than 3.5% of the total number of requests as shown in TABLE 5. It can improves the efficiency of SCCN furtherly.

7 Related Work

The existing research efforts of network caching technology can be divided into two categories: isolated cache replacement and multi-cache placement.

Isolated Cache Replacement: The three most common types of isolated cache are proxy server, standalone cache and browser-based cache. Studies about isolated cache mostly focus on the replacement strategy, which refers to the process that takes place when the cache becomes full and old contents must be removed to make space for new ones [15]. Replacement strategies can be classified into recency-based strategies and frequency-based strategies [15]. Recency-based strategies evict the object which was requested the least recently (e.g., LRU, LRU-Min). Frequency-based strategies evict the object which was accessed least frequently (e.g., LFU, LFU-DA). Some strategies mix recency and frequency and should be treated as recency/frequency-based strategies.

Multi-Cache Placement: Multi-cache paradigms mainly include hierarchical cache, distributed cache and hybrid cache. Hierarchical Web cache was first proposed in the Harvest project [25]. Similar content placement strategies are proposed in [30]–[32], which aim to minimize average access costs. However, in hierarchical architecture, high level caches may become bottlenecks. Distributed cache schemes only deploy caches at the bottom level. Several distributed cache systems were proposed in [33], [34], focusing on optimizing the transmission delay and eliminating transmission redundancy in mobile environments. Nevertheless, a large-scale deployment of distributed cache may encounter several problems such as higher connection time, higher bandwidth usage, etc. In a hybrid scheme, caches may cooperate with other caches at the same level or at a higher level. Applegate et al. [22] consider arbitrary networks with diverse disk and link bandwidth constraints. They employ a Lagrangian relaxation-based decomposition technique to find a near-optimal solution.

Contrary to existing solutions, our schemes employ centralized control plane to coordinate distributed cache nodes in networks. To alleviate the performance bottleneck of ISP networks, the placement decision aims to minimize the users' transmission delay while satisfying the disk space and access links' capacity constraints. SCCN Controller makes content placement decisions based on request history (frequency-based) and increment recording (recency-based), which can adapt easily to time varying changes in the access probabilities.

8 Conclusions

In this paper, we propose an SDN-based Cooperative Cache Network (SCCN), which can reduce the users' transmission delay and limit the inter-ISP traffic. An increment recording mechanism is designed, which can adapt to popularity change of contents. An approximation algorithm is proposed to solve the content placement problem with an approximation ratio of $1/2$ in the worst case. Furthermore, we design an efficient heuristic algorithm, which can not only allow the Controller to update the placement more frequently, but also achieve a tradeoff between the transmission delay and Internet traffic. Extensive trace-based simulation and experimental results demonstrate the effectiveness of SCCN under various network conditions.

Acknowledgement

This work is supported by the National High Technology Research and Development Program(”863”Program) of China (no.2015AA016101), Tsinghua National Laboratory for Information Science and Technology (TNList), Tsinghua University Initiative Scientific Research Program and the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CityU 117913) and the Science and Technology Project of State Grid Corporation of China(No. SGRIXTJSFW[2016]377).

References

[1] “Cisco visual networking index: Forecast and methodology, 2014-2019,” *Cisco Public Information*, 2015.
[2] V. Pacifici, F. Lehrieder, and G. Dán, “Cache capacity allocation for bittorrent-like systems to minimize inter-isp traffic,” in *IEEE INFOCOM*, 2012.
[3] Y. Huang, T. Z. Fu, D.-M. Chiu, J. Lui, and C. Huang, “Challenges, design and analysis of a large-scale p2p-vod system,” in *ACM SIGCOMM*, 2008.
[4] S. Borst, V. Gupta, and A. Walid, “Distributed caching algorithms for content distribution networks,” in *IEEE INFOCOM*, 2010.
[5] M. Qian, Y. Wang, Y. Zhou, L. Tian, and J. Shi, “A super base station based centralized network architecture for 5g mobile communication systems,” *Digital Communications and Networks*, vol. 1, no. 2, pp. 152–159, 2015.
[6] F. Chen, K. Guo, J. Lin, and T. La Porta, “Intra-cloud lightning: Building cdns in the cloud,” in *IEEE INFOCOM*, 2012.
[7] D. Ciullo, V. Martina, M. Garetto, and E. Leonardi, “How much can large-scale video-on-demand benefit from users' cooperation?” in *IEEE INFOCOM*, 2013.
[8] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber, “Pushing cdn-isp collaboration to the limit,” *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 3, pp. 34–44, 2013.
[9] D. J. McLaggan, “Web cache communication protocol (wccp) (version 2),” *IETF Draft*, 2012.
[10] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang, “A case for a coordinated internet video control plane,” in *ACM SIGCOMM*, 2012.
[11] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang, “Practical, real-time centralized control for cdn-based live video delivery,” in *ACM SIGCOMM*, 2015.
[12] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, “Openflow: enabling innovation in campus networks,” in *ACM SIGCOMM*, 2008.
[13] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, “Logically centralized?: state distribution trade-offs in software defined networks,” in *ACM SIGCOMM HOT SDN*, 2012.

[14] Q. Duan, "Modeling and performance analysis for composite network–compute service provisioning in software-defined cloud environments," *Digital Communications and Networks*, vol. 1, no. 3, pp. 181–190, 2015.

[15] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *Computing Surveys (CSUR)*, vol. 35, no. 4, 2003.

[16] D. B. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 62, no. 1-3, pp. 461–474, 1993.

[17] D. S. Hochbaum, "Heuristics for the fixed cost median problem," *Mathematical programming*, vol. 22, no. 1, pp. 148–162, 1982.

[18] "Open vswitch," http://openvswitch.org/.

[19] "Ryu," http://osrg.github.com/ryu/.

[20] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Web caching on smartphones: ideal vs. reality," in *ACM MobiSys*, 2012.

[21] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and zipf-like distributions: Evidence and implications," in *IEEE INFOCOM*, 1999.

[22] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale vod system," in *ACM CoNEXT*, 2010.

[23] B. Ager, F. Schneider, J. Kim, and A. Feldmann, "Revisiting cacheability in times of user generated content," in *IEEE INFOCOM*, 2010.

[24] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *ACM SIGMETRICS*, 2009.

[25] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell, "A hierarchical internet object cache," *Usenix*, 1996.

[26] "youku," http://www.youku.com/.

[27] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *ACM HotNets*, 2010.

[28] "mitmproxy," https://mitmproxy.org/.

[29] "Django," https://docs.djangoproject.com/en/1.8/.

[30] W. Li, E. Chan, G. Feng, D. Chen, and S. Lu, "Analysis and performance study for coordinated hierarchical cache placement strategies," *Computer Communications*, vol. 33, no. 15, pp. 1834–1842, 2010.

[31] Y. Kim and I. Yeom, "Performance analysis of in-network caching for content-centric networking," *Computer Networks*, vol. 57, no. 13, pp. 2465–2482, 2013.

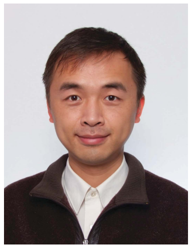
[32] M. Mangili, F. Martignon, and A. Capone, "A comparative study of content-centric and content-distribution networks: Performance and bounds," in *IEEE GLOBECOM*, 2013.

[33] S. Sanadhya, R. Sivakumar, K.-H. Kim, P. Congdon, S. Lakshmanan, and J. P. Singh, "Asymmetric caching: improved network deduplication for mobile devices," in *ACM MOBICOM*, 2012.

[34] S.-H. Shen and A. Akella, "An information-aware qoe-centric mobile video cache," in *ACM MOBICOM*, 2013.

Yong Cui received the B.E. degree and the Ph.D. degree in Computer Science and Engineering from Tsinghua University, China, in 1999 and 2004, respectively. He is currently a full professor in Tsinghua University, Co-Chair of IETF IPv6 Transition WG Softwire. Having published more than 100 papers in refereed journals and conferences, he received the National Award for Technological Invention in 2013, the Influential Invention Award of China Information Industry in both 2012 and 2004. He authored 3 Internet standard documents, including RFC 7040 and RFC 5565, for his proposal on IPv6 transition technologies. He serves at the Editorial Board on both IEEE TPDS and IEEE TCC. His major research interests include mobile wireless Internet and computer network architecture.

Jian Song received the B.E and M.E degrees in Department of Computer Science and Technology from Information Engineering University, Zhengzhou, China, in 2003 and 2006, respectively. He is currently working towards his Ph.D. degree in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests are in the areas of wireless networking and mobile cloud computing.



Minming Li received the B.E and PhD degrees in Department of Computer Science and Technology from Tsinghua University, Beijing, China, in 2002 and 2006, respectively. He is currently an associate professor in the Department of Computer Science, City University of Hong Kong. His research interests include wireless ad hoc networks, algorithm design and analysis, and combinatorial optimization.

Qingmei Ren received the B.E degrees in Department of Computer Science and Technology from Beijing University of Posts and Telecommunications, Beijing, China, in 2015. He is currently working towards his Master degree in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. Her research interests are in the areas of wireless networking and mobile cloud computing.

Yangjun Zhang received the B.E degrees in Department of Computer Science and Technology from Beijing University of Posts and Telecommunications, Beijing, China, in 2013. He is currently working towards his Master degree in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His research interests are in the areas of wireless networking and mobile cloud computing.

Xuejun Cai received his Ph.D. degree from Chinese Academy of Sciences, Beijing, China, in 2000. Since 2009, he has been with Ericsson Research, and his research interests includes content delivery, wireless network, Internet of things, cloud management. He has authored multiple journal articles, conference papers, patent applications and has been involved in 3GPP and IETF standardization.